



`</>` htmx & Web Components

A Perfect Match for Frontend Development

Horacio Gonzalez

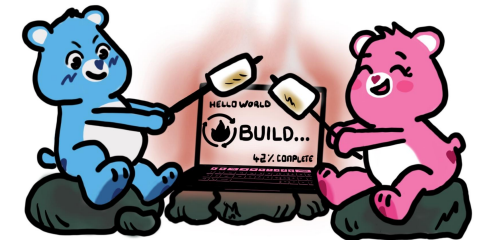
@LostInBrittany

Espagnol Perdu en Bretagne

Head of DevRel



clever cloud



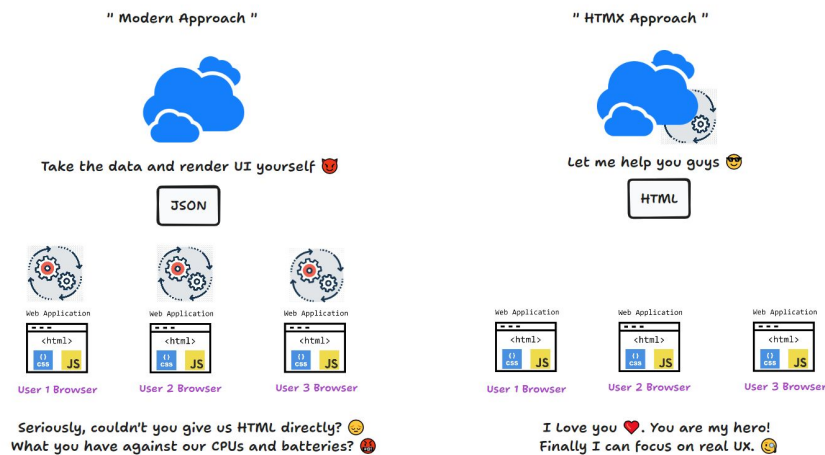
clever cloud



@LostInBrittany

`</> htmx`

Create modern user interfaces with
the simplicity and power of hypertext



Arbitrary Limitations of HTML



- Why can only `<a>` and `<form>` make HTTP(S) requests?
- Why can only click and submit events trigger them?
- Why are only `GET` and `POST` methods available?
- Why do `<a>` and `<form>` force a full page reload?
- Why so many arbitrary constraints in HTML?



Goal: Interactivity in Hypertext



htmx extends HTML capabilities to:

- Perform AJAX requests
- Handle CSS transitions
- Work with WebSockets
- Process server-sent events

All through declarative HTML attributes

`</> htmx`

But, what's the point?

It sounds nice and semantic, but what's the real benefit?



A Quick Look Back

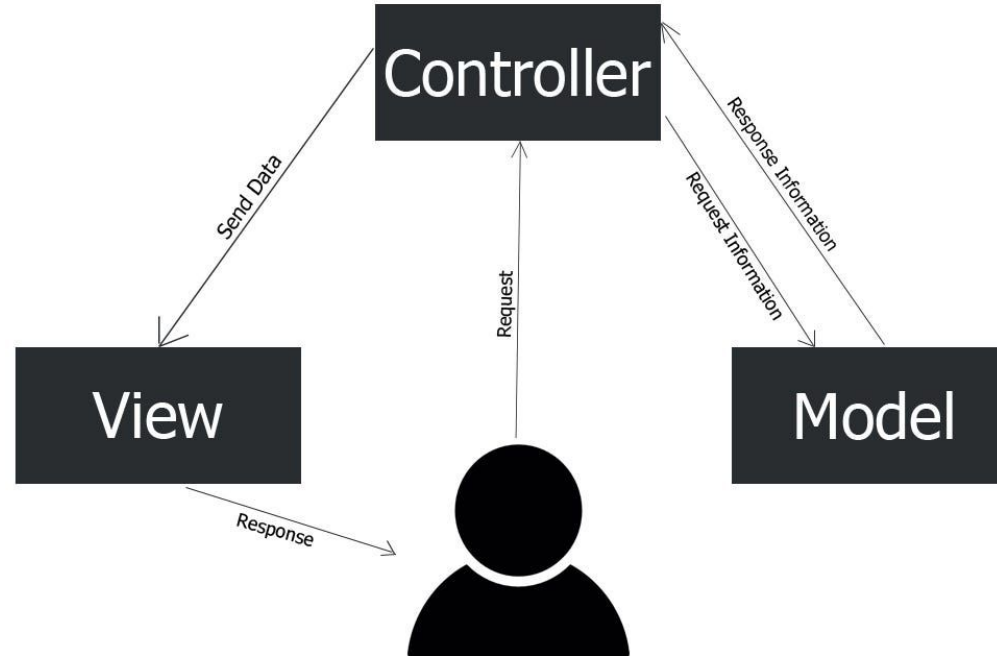


A time when dinosaurs like me coded with Struts



Remember MVC?

Model-View-Controller



With its page-by-page navigation

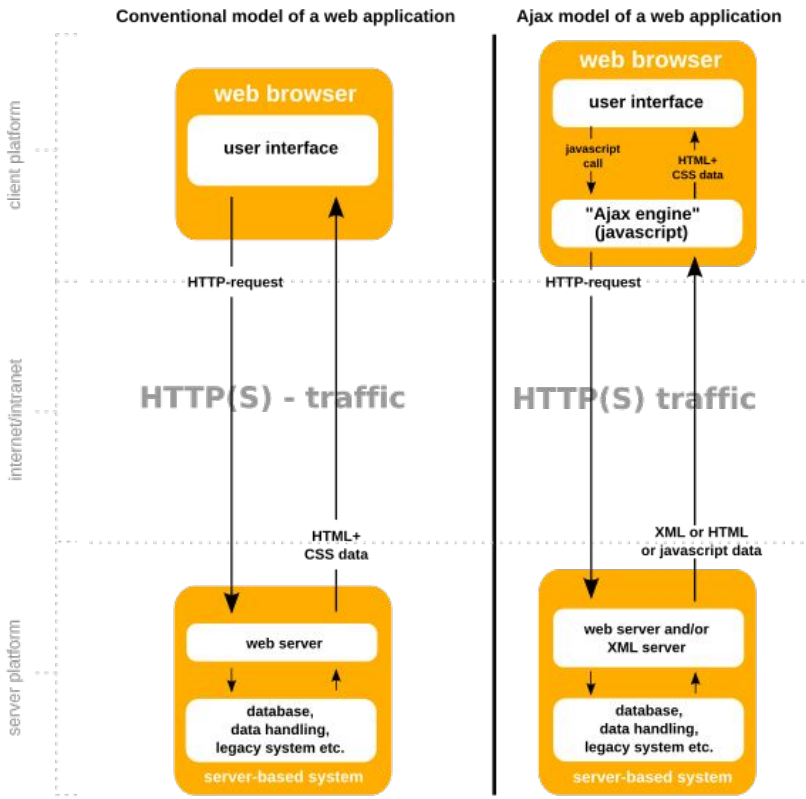
The Golden Age of MVC Frameworks



Generating HTML views



2005: The Arrival of AJAX



The birth of Web 2.0

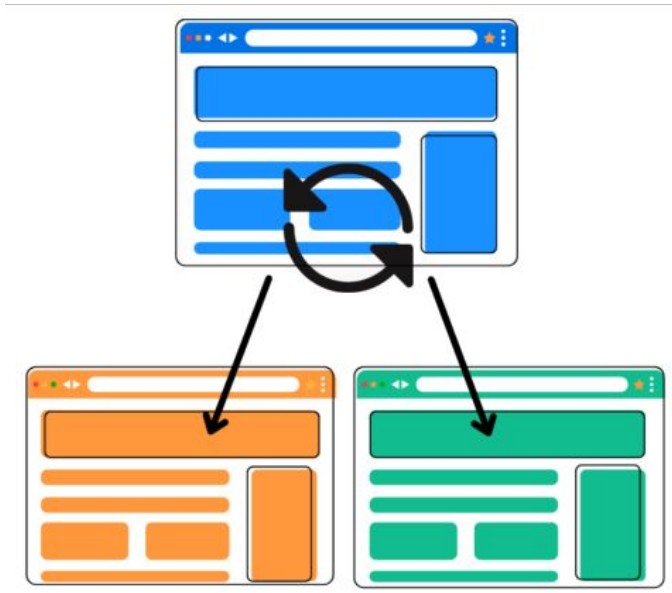
Web Pages Become Dynamic Apps



Powered by JavaScript and jQuery



Shift to Single Page Applications (SPA)

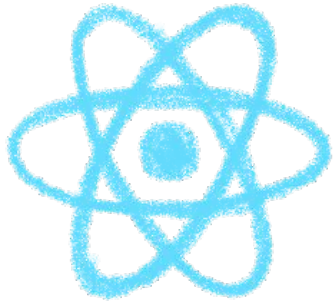


MPA
Multi-page app

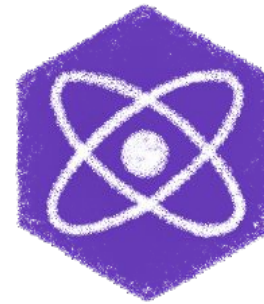


SPA
Single-page app

Increasing Complexity



Lit

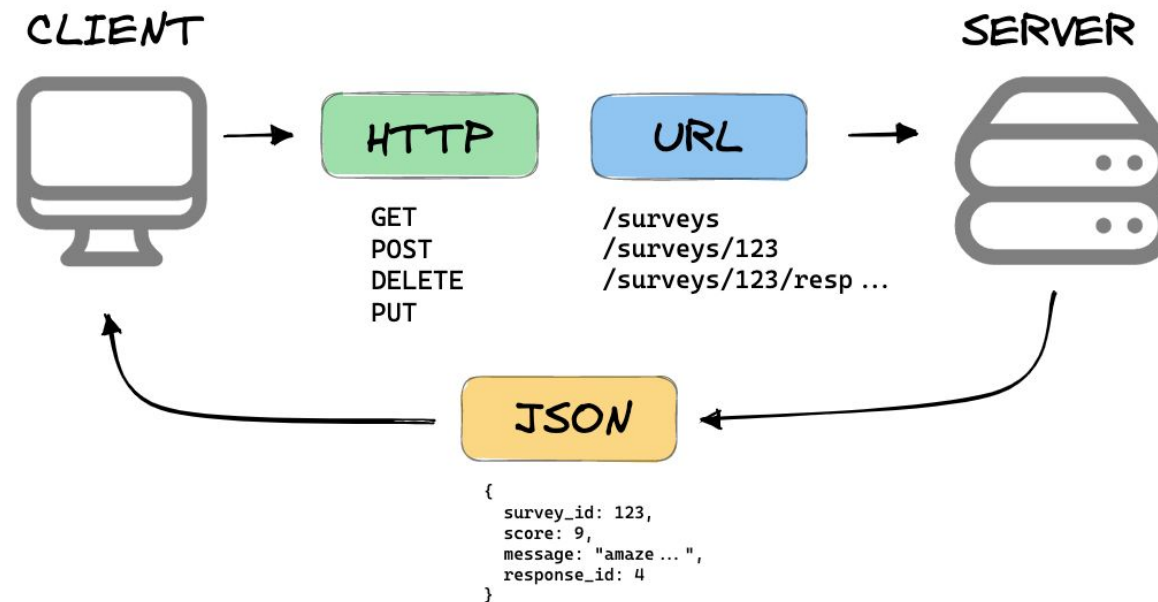


The rise of JavaScript frameworks



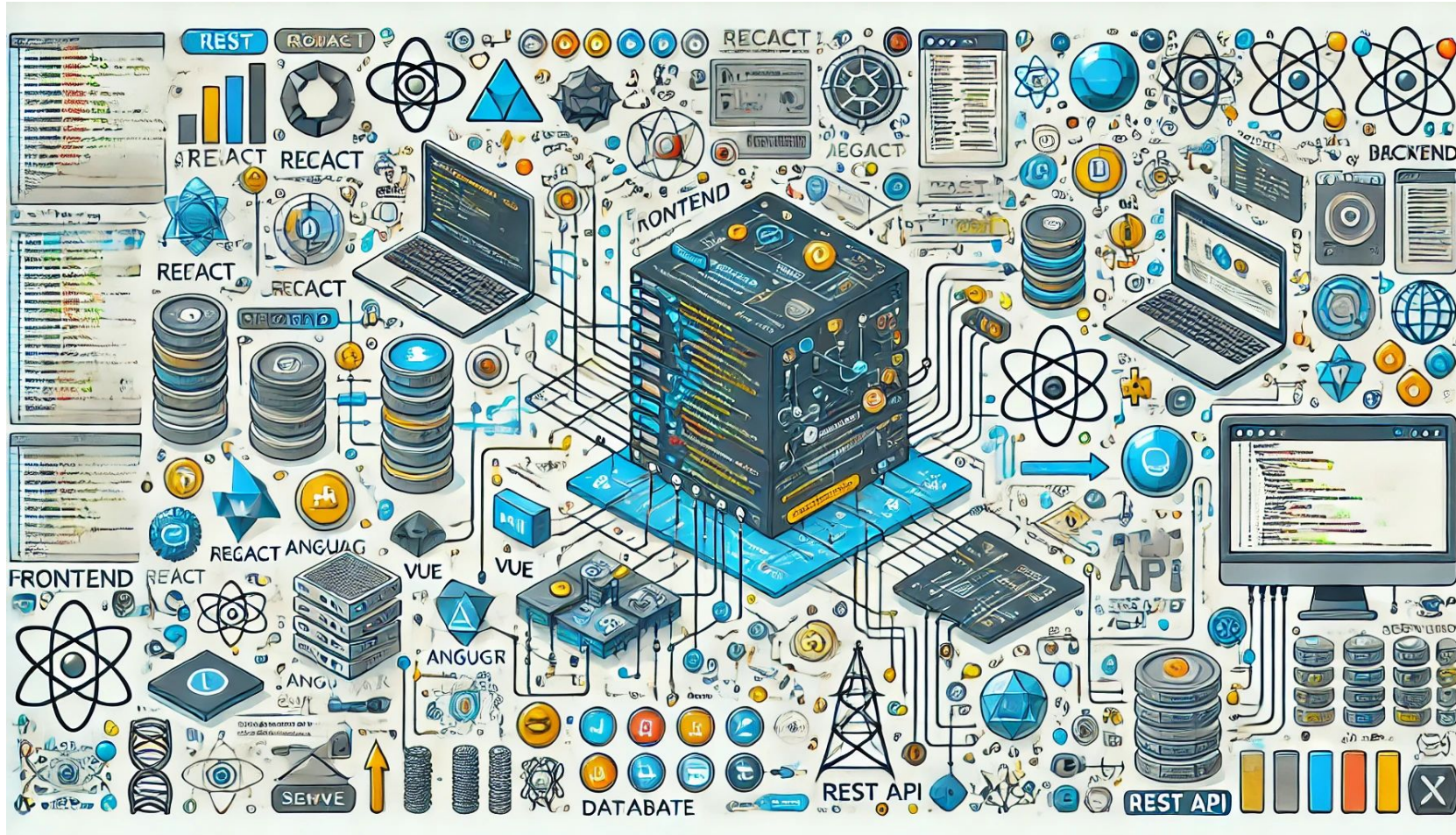
Backend Becomes a REST API

WHAT IS A REST API?



Serving JSON

We Gained Functionality



But lost simplicity and semantics



clever cloud

@LostInBrittany



Overkill for Many Applications



Sometimes we just need a simple web page
with a bit of interactivity

</> htmx Might Be the Right Solution



</> htmx

It's extended HTML

- Simplicity
- Semantics
- Interactivity

" Modern Approach "



Take the data and render UI yourself 🤖

JSON



Web Application



User 1 Browser



Web Application



User 2 Browser



Web Application



User 3 Browser

Seriously, couldn't you give us HTML directly? 😞
What you have against our CPUs and batteries? 🤖

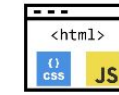
" HTMX Approach "



Let me help you guys 😊

HTML

Web Application



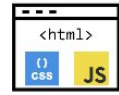
User 1 Browser

Web Application



User 2 Browser

Web Application



User 3 Browser

I Love you ❤️. You are my hero!
Finally I can focus on real UX. 😊



clever cloud

@LostInBrittany A cartoon character with a beard, glasses, and a blue shirt, holding a yellow pencil.

Too much theory, show us a demo!

Examples, examples, examples!

```
example-01.html
<script src="https://unpkg.com/htmx.org@2.0.2"></script>

<!-- have a button that POST on a click via AJAX
      and replace the content of #status div
      with the response -->

<button hx-post="/clicked" hx-target="#status">
  Click Me
</button>

<div id="status">Not yet clicked</div>
```

</> htmx



Too much theory, show us a demo!



`</>` htmx & Web Components: A Perfect Match for Frontend Development

This repository contains all the code and examples from my talk:

htmx 2.0 & Web Components: A Perfect Match for Frontend Development, presented at:

- 2024-09-24 - [FinistDevs](#) (Brest, France) - [Slides \(in French\)](#)
- 2025-02-04 - [Jfokus](#) (Stockholm, Sweden) - [Slides](#)
- 2025-04-16 - [Devoxx France](#) (Paris, France)
- 2025-06-05 - [DevQuest](#) (Niort, France)
- 2025-06-26 - [SunnyTech](#) (Montpellier, France)
- 2025-07-08 - [DevBcn](#) (Barcelona, Spain)

This talk explores how **htmx 2.0** enhances HTML with seamless interactivity while **Web Components** (Lit) encapsulate logic and styling, providing a powerful yet lightweight alternative to heavy frontend frameworks.



`</>` htmx & Web Components

A Perfect Match for Frontend Development



<https://github.com/LostInBrittany/introduction-to-htmx-and-lit>



clever cloud



Sending POST on a button click



```
./html-examples/html-example-01.html

<script src="https://unpkg.com/hmx.org@2.0.2"></script>

<!-- have a button that POST on a click via AJAX
      and replace the content of #status div
      with the response -->

<button hx-post="/clicked" hx-target="#status">
  Click Me
</button>

<div id="status">Not yet clicked</div>
```

</> htmx

GET, POST, PUT, DELETE...



```
./html-examples/html-example-02.html

<script src="https://unpkg.com/htmx.org@2.0.2"></script>

<div>
  <button hx-get="/test-methods" hx-target="#status">Send GET</button>
  <button hx-post="/test-methods" hx-target="#status">Send POST</button>
  <button hx-put="/test-methods" hx-target="#status">Send PUT</button>
  <button hx-delete="/test-methods" hx-target="#status">Send DELETE</button>
</div>

<div id="status">No request sent</div>
```

</> htmx

Using response to replace elements



```
./html-examples/html-example-03.html

<script src="https://unpkg.com/htmx.org@2.0.2"></script>

<div id="test-replace">
  <button hx-get="/test-replace/innerHTML">
    If you click, this message will be replaced
  </button>
  <button hx-get="/test-replace/outerHTML" hx-swap="outerHTML">
    If you click, this button will become a div
  </button>
  <button hx-get="/test-replace/delete" hx-swap="delete">
    If you click, this button will disappear when the response is received
  </button>
  <button hx-get="/test-replace/none" hx-swap="none">
    If you click, nothing changes, the response is ignored
  </button>
</div>
```

</> htmx



Choosing when to send requests



```
./html-examples/html-example-04.html

<script src="https://unpkg.com/htmx.org@2.0.2"></script>

<!-- By default, AJAX requests are triggered by the "natural" event of an element: -->
<div id="test-triggers">
  <button hx-get="/trigger/natural" hx-target="#status">
    In a button the natural event is a click
  </button>
  <button hx-trigger="mouseover" hx-get="/trigger/mouseover" hx-target="#status">
    This button triggers on mouseover
  </button>
  <button hx-trigger="mouseenter" hx-get="/trigger/mouseenter" hx-target="#status">
    This button triggers on mouseenter
  </button>
  <button hx-trigger="mouseleave" hx-get="/trigger/mouseleave" hx-target="#status">
    This button triggers on mouseleave
  </button>
</div>

<div id="status">No AJAX request sent yet</div>
```

</> htmx



More triggering options



```
./html-examples/html-example-05.html

<script src="https://unpkg.com/htmx.org@2.0.2"></script>

<!-- By default, AJAX requests are triggered by the "natural" event of an element: -->
<div id="test-triggers">
  <button hx-trigger="every 5s" hx-get="/trigger/5seconds" hx-target="#status">
    Sends request every 5 seconds, no event needed
  </button>
  <button hx-trigger="click[ctrlKey]" hx-get="/trigger/ctrlclick" hx-target="#status">
    Sends request on click while pressing Ctrl
  </button>
  <button hx-trigger="click[ctrlKey] once" hx-get="/trigger/ctrlclickonce" hx-target="#status">
    Sends request on the first click while pressing Ctrl
  </button>
</div>

<div id="status">No AJAX request sent yet</div>
```

</> htmx



A spinner to ease you wait



```
./html-examples/html-example-06.html

<script src="https://unpkg.com/htmx.org@2.0.2"></script>

<!-- By default, AJAX requests are triggered by the "natural" event of an element: -->
<div id="test-triggers">
  <button hx-trigger="every 5s" hx-get="/trigger/5seconds" hx-target="#status">
    Sends request every 5 seconds, no event needed
  </button>
  <button hx-trigger="click[ctrlKey]" hx-get="/trigger/ctrlclick" hx-target="#status">
    Sends request on click while pressing Ctrl
  </button>
  <button hx-trigger="click[ctrlKey] once" hx-get="/trigger/ctrlclickonce" hx-target="#status">
    Sends request on the first click while pressing Ctrl
  </button>
</div>

<div id="status">No AJAX request sent yet</div>
```

</> htmx



Extensions, extensions everywhere



</> htmx extensions

This site is a searchable collection of extensions for [htmx 2.0](#). They are not guaranteed to work with the htmx 1.x codebase.

[Core](#) extensions are actively maintained by the htmx team.

[Community](#) extensions are contributed by the community or rarely touched by the htmx team (although they still work!)

► Contributing

[Core](#)

Name	Description
sse	Provides support for Server Sent Events directly from HTML.
ws	Provides bi-directional communication with Web Sockets servers directly from HTML

Time for More Code!

Let's see a complete example

To-do example

- Think about tasks



Too much theory, show us a demo!



`</>` htmx & Web Components: A Perfect Match for Frontend Development

This repository contains all the code and examples from my talk:

htmx 2.0 & Web Components: A Perfect Match for Frontend Development, presented at:

- 2024-09-24 - [FinistDevs](#) (Brest, France) - [Slides \(in French\)](#)
- 2025-02-04 - [Jfokus](#) (Stockholm, Sweden) - [Slides](#)
- 2025-04-16 - [Devoxx France](#) (Paris, France)
- 2025-06-05 - [DevQuest](#) (Niort, France)
- 2025-06-26 - [SunnyTech](#) (Montpellier, France)
- 2025-07-08 - [DevBcn](#) (Barcelona, Spain)

This talk explores how **htmx 2.0** enhances HTML with seamless interactivity while **Web Components** (Lit) encapsulate logic and styling, providing a powerful yet lightweight alternative to heavy frontend frameworks.



`</>` htmx & Web Components

A Perfect Match for Frontend Development



<https://github.com/LostInBrittany/introduction-to-htmx-and-lit>



clever cloud



Let's do a to-do list



To-do example

- Think about tasks

addTask

From Hello World to a To-do List

What the heck are web component?

The 3 minutes context



Web Components



Web standard W3C

Web Components



Available in all modern browsers:
Firefox, Safari, Chrome



Create your own HTML tags
Encapsulating look and behavior



Fully interoperable

With other web components, with any framework

Web Components



CUSTOM ELEMENTS



SHADOW DOM



TEMPLATES



clever cloud

@LostInBrittany



Custom Element



To define your own HTML tag

```
<body>
  ...
  <script>
    window.customElements.define('my-element',
      class extends HTMLElement {...});
  </script>
  <my-element></my-element>
</body>
```



Shadow DOM



To encapsulate subtree and style in an element

```
<button>Hello, world!</button>
```

```
<script>
```

```
var host = document.querySelector('button');
```

```
const shadowRoot = host.attachShadow({mode: 'open'});
```

```
shadowRoot.textContent = 'こんにちは、影の世界!';
```

```
</script>
```

Hello, world!



こんにちは、影の世界!



clever cloud



To have clonable document template

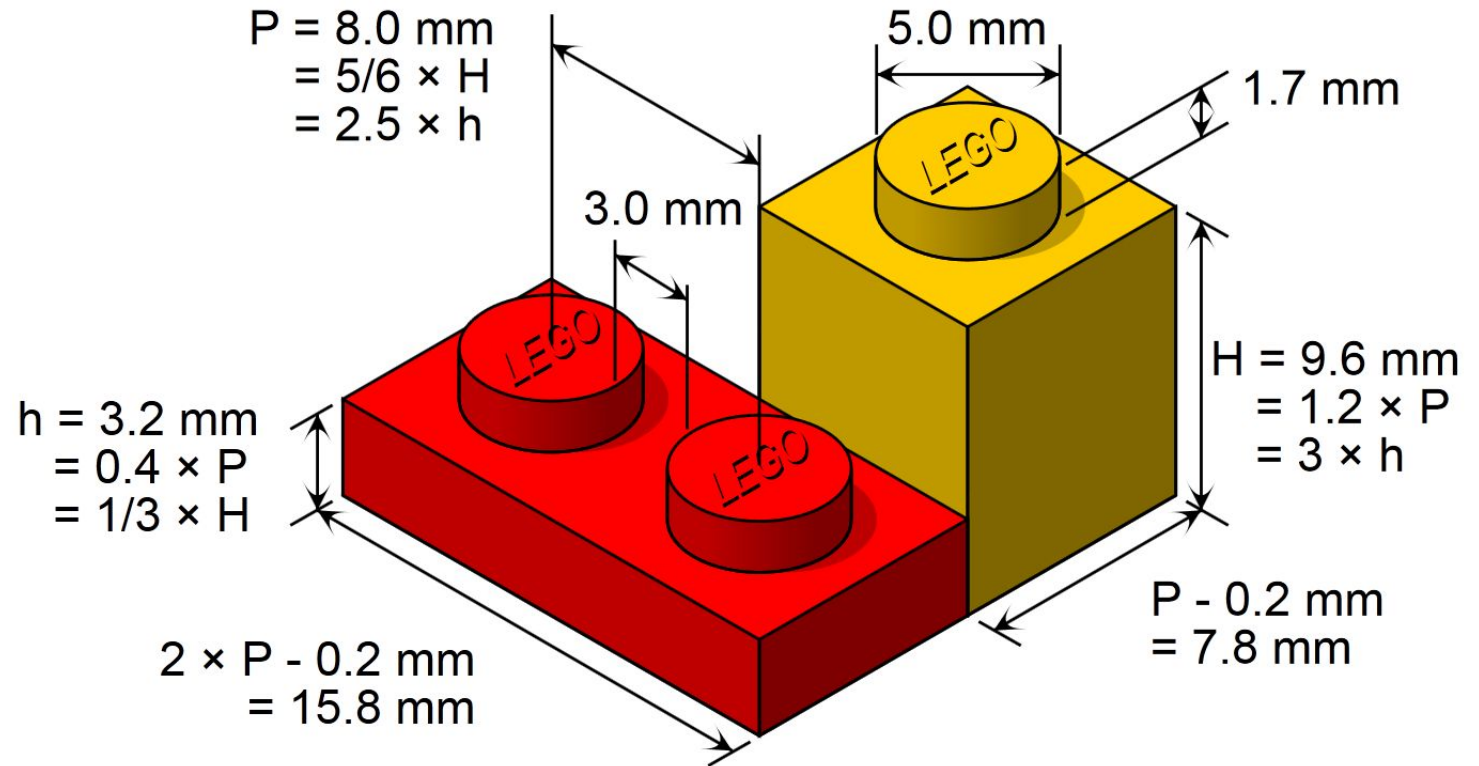
```
<template id="mytemplate">  
  <img src="" alt="great image">  
  <div class="comment"></div>  
</template>
```

```
var t = document.querySelector('#mytemplate');  
// Populate the src at runtime.  
t.content.querySelector('img').src = 'logo.png';  
var clone = document.importNode(t.content, true);  
document.body.appendChild(clone);
```



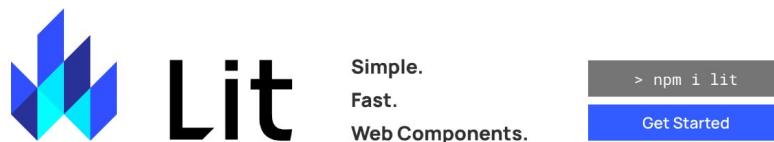
But in fact, it's just an element...

- Attributes
- Properties
- Methods
- Events





Simple. Fast. Web Components



Simple

Skip the boilerplate

Building on top of the Web Components standards, Lit adds just what you need to be happy and productive: reactivity, declarative templates and a handful of thoughtful features to reduce boilerplate and make your

Fast

Tiny footprint, instant updates

Weighing in at around 5 KB (minified and compressed), Lit helps keep your bundle size small and your loading time short. And rendering is blazing fast, because Lit touches only the dynamic parts of your UI when updating

Web Components

Interoperable & future-ready

Every Lit component is a native web component, with the superpower of interoperability. Web components work anywhere you use HTML, with any framework or none at all. This makes Lit ideal for building shareable

Modern lightweight web components



Lit

Simple.
Fast.
Web Components.

```
> npm i lit
```

Get Started



Simple

Skip the boilerplate

Building on top of the Web Components standards, Lit adds just what you need to be happy and productive: reactivity, declarative templates and a handful of thoughtful features to reduce boilerplate and make your



Fast

Tiny footprint, instant updates

Weighing in at around 5 KB (minified and compressed), Lit helps keep your bundle size small and your loading time short. And rendering is blazing fast, because Lit touches only the dynamic parts of your UI when updating



Web Components

Interoperable & future-ready

Every Lit component is a native web component, with the superpower of interoperability. Web components work anywhere you use HTML, with any framework or none at all. This makes Lit ideal for building shareable

For the new web paradigm



clever cloud



```
import { LitElement, html } from 'lit-element';

// Create your custom component
class CustomGreeting extends LitElement {
  // Declare properties
  static get properties() {
    return {
      name: { type: String }
    };
  }
  // Initialize properties
  constructor() {
    super();
    this.name = 'World';
  }
  // Define a template
  render() {
    return html`<p>Hello, ${this.name}</p>`;
  }
}
// Register the element with the browser
customElements.define('custom-greeting', CustomGreeting);
```

Lightweight web-components using lit-html

Based on lit-html



lit / packages / lit-html / README.md



abdonrd Unify the npm badges (#3680) ✕

d85f082 - 5 months ago History

Preview

Code

Blame

58 lines (36 loc) · 2.7 KB

Raw



lit-html 2.0

Efficient, Expressive, Extensible HTML templates in JavaScript



lit-html is the template system that powers the [Lit](#) library for building fast web components. When using `lit-html` to develop web components, most users should import lit-html via the `lit` package rather than installing and importing from `lit-html` directly.

About this release

This is a stable release of `lit-html` 2.0 (part of the Lit 2.0 release). If upgrading from previous versions of `lit-html`, please note the minor breaking changes from lit-html 1.0 in the [Upgrade Guide](#).

An efficient, expressive, extensible
HTML templating library for JavaScript



clever cloud



Do you know tagged templates?



```
function uppercaseExpression(strings, ...expressionValues) {  
  var finalString = ''  
  for ( let i = 0; i < strings.length; i++ ) {  
    if (i > 0) {  
      finalString += expressionValues[i - 1].toUpperCase()  
    }  
    finalString += strings[i]  
  }  
  return finalString  
}  
  
const expressions = [ 'Sophia Antipolis', 'RivieraDev', 'Thank you'];  
console.log(uppercase`Je suis à ${expression[0]} pour ${expression[1]}. ${expression[2]}!`
```

Little known functionality of template literals



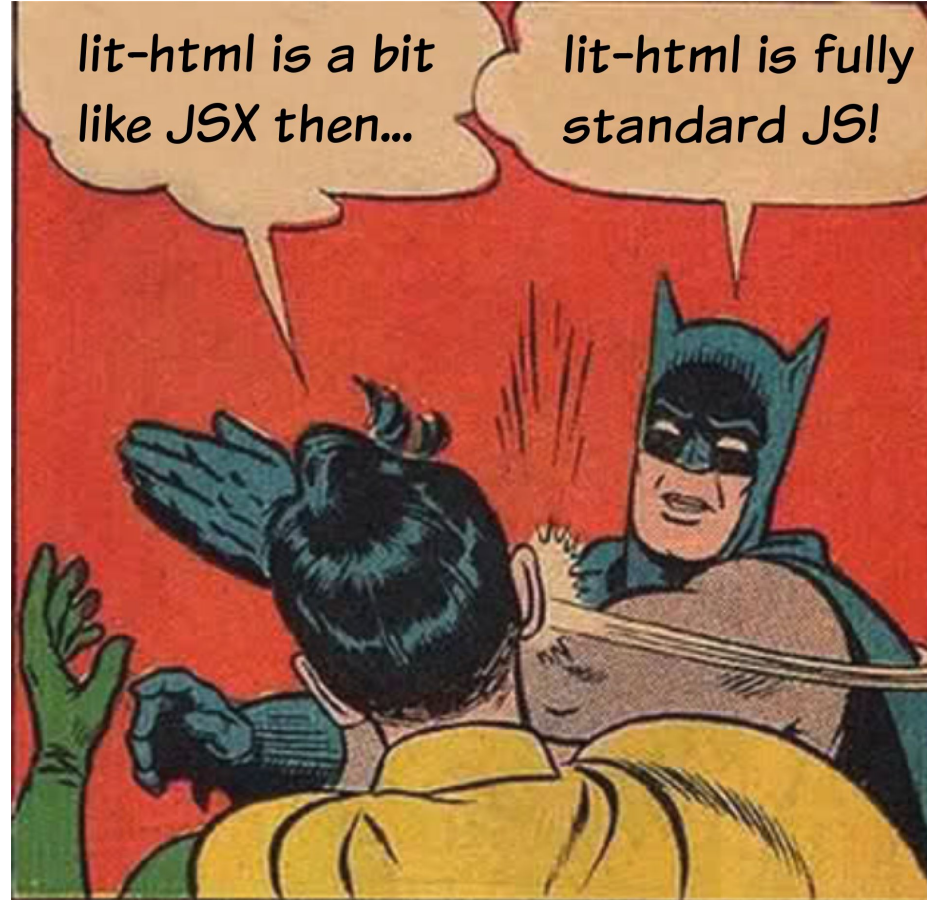
clever cloud



```
let myTemplate = (data) => html`  
  <h1>${data.title}</h1>  
  <p>${data.body}</p>  
`;  
;
```

Lazily rendered
Generates a TemplateResult

It's a bit like JSX, isn't it?



The good sides of JSX... but in the standard!

Too much theory, show us a demo!



`</>` htmx & Web Components: A Perfect Match for Frontend Development

This repository contains all the code and examples from my talk:

htmx 2.0 & Web Components: A Perfect Match for Frontend Development, presented at:

- 2024-09-24 - [FinistDevs](#) (Brest, France) - [Slides \(in French\)](#)
- 2025-02-04 - [Jfokus](#) (Stockholm, Sweden) - [Slides](#)
- 2025-04-16 - [Devoxx France](#) (Paris, France)
- 2025-06-05 - [DevQuest](#) (Niort, France)
- 2025-06-26 - [SunnyTech](#) (Montpellier, France)
- 2025-07-08 - [DevBcn](#) (Barcelona, Spain)

This talk explores how **htmx 2.0** enhances HTML with seamless interactivity while **Web Components** (Lit) encapsulate logic and styling, providing a powerful yet lightweight alternative to heavy frontend frameworks.



`</>` htmx & Web Components

A Perfect Match for Frontend Development



<https://github.com/LostInBrittany/introduction-to-htmx-and-lit>



clever cloud



Custom Greeting example



```
import { LitElement, html } from 'lit-element';

// Create your custom component
class CustomGreeting extends LitElement {
  // Declare properties
  static get properties() {
    return {
      name: { type: String }
    };
  }
  // Initialize properties
  constructor() {
    super();
    this.name = 'World';
  }
  // Define a template
  render() {
    return html`<p>Hello, ${this.name}!</p>`;
  }
}
// Register the element with the browser
customElements.define('custom-greeting', CustomGreeting);
```

Lightweight web-components using lit-html



clever cloud



My Lit Counter example



Let's do an interactive counter

Lit & `</>` htmx

Love at first `<tag>`



Lit

`</>` htmx



htmx for structure, Lit to encapsulate logic



</> htmx



Lit

To htmx, Lit elements are just regular tags

Too much theory, show us a demo!



`</>` htmx & Web Components: A Perfect Match for Frontend Development

This repository contains all the code and examples from my talk:

htmx 2.0 & Web Components: A Perfect Match for Frontend Development, presented at:

- 2024-09-24 - [FinistDevs](#) (Brest, France) - [Slides \(in French\)](#)
- 2025-02-04 - [Jfokus](#) (Stockholm, Sweden) - [Slides](#)
- 2025-04-16 - [Devoxx France](#) (Paris, France)
- 2025-06-05 - [DevQuest](#) (Niort, France)
- 2025-06-26 - [SunnyTech](#) (Montpellier, France)
- 2025-07-08 - [DevBcn](#) (Barcelona, Spain)

This talk explores how **htmx 2.0** enhances HTML with seamless interactivity while **Web Components** (Lit) encapsulate logic and styling, providing a powerful yet lightweight alternative to heavy frontend frameworks.



`</>` htmx & Web Components

A Perfect Match for Frontend Development



<https://github.com/LostInBrittany/introduction-to-htmx-and-lit>



clever cloud



That's all, folks!

Thank you all!



*Please leave your
feedback!*



clever cloud

@LostInBrittany

